Computer Software: The 'Trojan Horse' of HPC

Co-Designing a COTS Re-configurable Exascale Computer

Steve Wallach
swallach”at”conveycomputer “dot”com
Convey <= Convex++
Acknowledgement

• Steve Poole (ORNL) and I had many emails and discussion on technology trends.

• We are both guilty of being overzealous architects
Discussion

- The state of HPC software – today
- The path to Exascale Computing
Mythology

- In the old days, we were told
  “Beware of Greeks bearing gifts”
Today

• “Beware of Geeks bearing Gifts”
What problems are we solving

- New Hardware Paradigms
- Uniprocessor Performance leveling
- MPP and multi-threading for the masses
Deja Vu

• Multi-Core Evolves
  – Many Core
  – ILP fizzes
• x86 extended with sse2, sse3, and sse4
  – application specific enhancements
  – Co-processor within x86 micro-architecture
• Basically performance enhancements by
  – On chip parallel
  – Instructions for specific application acceleration
    • One application instruction replaces MANY generic instructions
• Déjà vu – all over again – 1980’s
  – Need more performance than micro
  – GPU, CELL, and FPGA’s
    • Different software environment
• Heterogeneous Computing AGAIN
Current Languages

- Fortran 66 ➔ Fortran 77 ➔ Fortran 95 ➔ 2003
  - HPC Fortran
  - Co-Array Fortran
- C ➔ C++
  - UPC
  - Stream C
  - C# (Microsoft)
  - Ct (Intel)
Another Bump in the Road

- GPGPU’s are very cost effective for many applications.
- Matrix Multiply
  - Fortran

```fortran
  do i = 1,n1
    do k = 1,n3
      c(i,k) = 0.0
      do j = 1,n2
        c(i,k) = c(i,k) + a(i,j) * b(j,k)
      enddo
    enddo
  enddo
```

swallach – CARL – Micro 43
__global__ void 
matmulKernel( float* C, float* A, float* B, int N2, int N3 ){
    int bx = blockIdx.x, by = blockIdx.y;
    int tx = threadIdx.x, ty = threadIdx.y;
    int aFirst = 16 * by * N2;
    int bFirst = 16 * bx;
    float Csub = 0;
    for( int j = 0; j < N2; j += 16 ) {
        shared float Atile[16][16], Btile[16][16];
        Atile[ty][tx] = A[aFirst + j + N2 * ty + tx];
        Btile[ty][tx] = B[bFirst + j*N3 + b + N3 * ty + tx];
        __syncthreads();
        for( int k = 0; k < 16; ++k )
            Csub += Atile[ty][k] * Btile[k][tx];
        __syncthreads();
    }
    int c = N3 * 16 * by + 16 * bx;
    C[c + N3 * ty + tx] = Csub;
}

void 
matmul( float* A, float* B, float* C,
    size_t N1, size_t N2, size_t N3 ){
    void *devA, *devB, *devC;
    cudaSetDevice(0);
    cudaMalloc( &devA, N1*N2*sizeof(float) );
    cudaMalloc( &devB, N2*N3*sizeof(float) );
    cudaMalloc( &devC, N1*N3*sizeof(float) );
    cudaMemcpy( devA, A, N1*N2*sizeof(float), cudaMemcpyHostToDevice );
    dim3 threads( 16, 16 );
    dim3 grid( N1 / threads.x, N3 / threads.y);
    matmulKernel<<<grid, threads>>>( devC, devA, devB, N2, N3 );
    cudaMemcpy( devC, devC, N1*N3*sizeof(float), cudaMemcpyDeviceToHost );
    cudaFree( devA );
    cudaFree( devB );
    cudaFree( devC );
}
Accelerators can be beneficial. It isn’t “free” (like waiting for the next clock speed boost)

- Worst case - you will have to completely rethink your algorithms and/or data structures
- Performance tuning is still time consuming
- Don’t forget our long history of parallel computing...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>=</td>
<td>Java</td>
<td>18.033%</td>
<td>-2.11%</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>=</td>
<td>C</td>
<td>17.809%</td>
<td>+1.03%</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>=</td>
<td>C++</td>
<td>10.757%</td>
<td>+0.16%</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>=</td>
<td>PHP</td>
<td>8.934%</td>
<td>-0.74%</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>=</td>
<td>(Visual) Basic</td>
<td>5.868%</td>
<td>-2.07%</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>↑</td>
<td>C#</td>
<td>5.196%</td>
<td>+0.66%</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>↓</td>
<td>Python</td>
<td>4.266%</td>
<td>-0.49%</td>
<td>A</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>↑</td>
<td>Perl</td>
<td>3.200%</td>
<td>-0.71%</td>
<td>A</td>
</tr>
<tr>
<td>9</td>
<td>45</td>
<td>↑↑↑↑↑↑↑↑↑↑</td>
<td>Objective-C</td>
<td>2.469%</td>
<td>+2.35%</td>
<td>A</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>↑</td>
<td>Delphi</td>
<td>2.394%</td>
<td>+0.21%</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>↓↓↓↓↓↓↓↓↓</td>
<td>JavaScript</td>
<td>2.191%</td>
<td>-1.83%</td>
<td>A</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>↓↓↓↓↓↓↓↓↓</td>
<td>Ruby</td>
<td>2.070%</td>
<td>-0.56%</td>
<td>A</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>↓</td>
<td>PL/SQL</td>
<td>0.787%</td>
<td>-0.09%</td>
<td>A</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>=</td>
<td>SAS</td>
<td>0.703%</td>
<td>-0.06%</td>
<td>A-</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>=</td>
<td>Pascal</td>
<td>0.702%</td>
<td>-0.06%</td>
<td>A-</td>
</tr>
<tr>
<td>16</td>
<td>18</td>
<td>↑↑</td>
<td>Lisp/Scheme/Clojure</td>
<td>0.654%</td>
<td>+0.05%</td>
<td>B</td>
</tr>
<tr>
<td>17</td>
<td>19</td>
<td>↑↑↑↑↑↑↑↑↑↑↑</td>
<td>Lua</td>
<td>0.592%</td>
<td>+0.04%</td>
<td>B</td>
</tr>
<tr>
<td>18</td>
<td>20</td>
<td>↑↑↑↑↑↑↑↑↑↑</td>
<td>MATLAB</td>
<td>0.589%</td>
<td>+0.06%</td>
<td>B</td>
</tr>
<tr>
<td>19</td>
<td>16</td>
<td>↓↓↓↓↓↓↓↓↓</td>
<td>ABAP</td>
<td>0.577%</td>
<td>-0.15%</td>
<td>B</td>
</tr>
<tr>
<td>20</td>
<td>27</td>
<td>↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑</td>
<td>PowerShell</td>
<td>0.529%</td>
<td>+0.23%</td>
<td>B</td>
</tr>
</tbody>
</table>
Cautionary Tale: A Brief History of Languages

- When vector machines were king
  - Parallel “languages” were loop annotations (IVDEP)
  - Performance was fragile, but there was good user support

Kathy Yelick, 2008 Keynote, Salishan Conference
Technology Roadmap

- Improved interconnect technology
- Improved hardware and software reliability
- New IO technology
- 3D chip-level integration
- New programming model
- Demonstrate > 3X power efficiency gain
- SW scalability to 100M threads
- Latency tolerant algorithms
- Demonstrate > 3X power efficiency gain over 2015
- 10X memory BW
- Application scalability to 1B threads
- Improved resilience through local recovery and migration
- Exascale Science
<table>
<thead>
<tr>
<th>Systems</th>
<th>2009</th>
<th>2018 Swimlane 1</th>
<th>2018 SwimLane 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>System peak</td>
<td>2 Petas</td>
<td>1 Exa</td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>6 MW</td>
<td>~20 MW</td>
<td></td>
</tr>
<tr>
<td>System memory</td>
<td>0.3 PB</td>
<td>50 PB</td>
<td></td>
</tr>
<tr>
<td>Node performance</td>
<td>1/2 Uf</td>
<td>1Uf</td>
<td>1Uf</td>
</tr>
<tr>
<td>Interconnect latency (for longest path)</td>
<td>1-5 usec (limited by overhead at endpoints)</td>
<td>0.5 usec (speed of light)</td>
<td></td>
</tr>
<tr>
<td>Memory latency</td>
<td>150-250 clock cycles (~70-100ns)</td>
<td>100 clock cycles (~50ns)</td>
<td></td>
</tr>
<tr>
<td>Node memory BW</td>
<td>25 GB/s</td>
<td>0.4TB/s</td>
<td>4-5TB/s</td>
</tr>
<tr>
<td>Node concurrency</td>
<td>12</td>
<td>O(1k)</td>
<td>O(10k)</td>
</tr>
<tr>
<td>Total Node Interconnect BW</td>
<td>3.5 GB/s</td>
<td>200GB/s</td>
<td>2TB/s</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>18,700</td>
<td>O(1M)</td>
<td>O(100,000)</td>
</tr>
<tr>
<td>Total concurrency</td>
<td>225,000</td>
<td>O(100M)*10 for latency hiding</td>
<td>O(100M)*100 for latency hiding</td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
<td>1000 PB (&gt;10x system memory is min)</td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>0.2 TB</td>
<td></td>
<td>60 TB/s</td>
</tr>
<tr>
<td>MTADT (mean time between application failures)</td>
<td>Days</td>
<td>34 hours</td>
<td></td>
</tr>
</tbody>
</table>
Berkeley’s 13 Motifs

“Motif/Dwarf” Popularity
(RED HOT → BLUE COOL)

- How do compelling apps relate to 13 motif/dwarfs?

http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-23.html
What does this all mean?

How do we create architectures to do this?

Each application is a different point on this 3D grid (actually a curve).
Take an Operation Research Method of Prediction

- Moore’s Law
- Application Specific
  - Matrices
  - Matrix Arithmetic
- Hard IP for floating point
- Number of Arithmetic Engines
- System Arch and Programming model
  - Language directed design
- Resulting Analysis
  - Benefit
  - Mean and +/- sigma (if normal)

Bayesian Decision Theory

In a formal model the conclusions are derived from definitions and assumptions. . . . But with informal, verbal reasoning. . . . one can argue until one is blue in the face . . . because there is no criterion for deciding the soundness of an informal argument.

Robert Aumann
Moore’s Law

• Feature Set
  – Every 2 years twice the logic
  – Thus by 2020
    • 8 times the logic, same clock rate
  – Mean Factor of 7, sigma +/- 2

• Benefit
  – 7 times the performance, same clock rate, same internal architecture
Rock’s Law

- **Rock's law**, named for Arthur Rock, says that the cost of a semiconductor chip **fabrication plant** doubles every four years. As of 2003, the price had already reached about 3 billion US dollars.

- Rock's Law can be seen as the economic flipside to **Moore's Law**; the latter is a direct consequence of the ongoing growth of the capital-intensive semiconductor industry—innovative and popular products mean more profits, meaning more capital available to invest in ever higher levels of **large-scale integration**, which in turn leads to creation of even more innovative products.

http://en.wikipedia.org/wiki/Rock%27s_law
Application Specific

- **Feature Set**
  - Matrix Engine
  - Mean Factor of 4 (+4/-1)
- **June 1993 Linpack (% of peak)**
  - NEC SX3/44 - 95% (4)
  - Cray YMP (16) – 90% (9)
- **June 2006**
  - Earth Simulator (NEC) – 87.5% (5120)
- **June 2010 Linpack**
  - Jaguar - 75% (224162 cores & GPU)
- **November 2010 Linpack**
  - Tianhe-1A – 53% (86016 cores & GPU)
  - French Bull – 83% (17,480 sockets. 140k cores)
- **Benefit**
  - 90% of Peak
  - Matrix Arithmetic’s
  - Outer Loop Parallel/Inner Loop vector within ONE functional unit
3D

- Modular content
  - Datatypes: IEEE SP, IEEE DP, 32-bit integer, 64-bit integer, bit
  - Operations: simple, compound, reductions
  - Dimensionality: 1d, 2d, 3d
  - Memory access: gather/scatter, strided, under mask, multidimensional
  - Register access pattern: halo, sparse periodic access, under mask, multidimensional
- Number of function units determined by functionality selected
3D Finite Difference (3DFD) Personality

• designed for nearest neighbor operations on structured grids
  – maximizes data reuse
• reconfigurable “registers”
  – 1D (vector), 2D, and 3D modes
  – 8192 elements in a register
• operations on entire cubes
  – “add points to their neighbor to the left times a scalar” is a single instruction
  – up to 7 points away in any direction
• finite difference method for post-stack reverse-time migration

\[ X(I,J,K) = S0 \cdot Y(I,J,K) + S1 \cdot Y(I-1,J,K) + S2 \cdot Y(I+1,J,K) + S3 \cdot Y(I,J-1,K) + S4 \cdot Y(I,J+1,K) + S5 \cdot Y(I,J,K-1) + S6 \cdot Y(I,J,K+1) \]
FPGA - Hard IP Floating Point

• Feature Set
  – By 2020
  – Fused Multiply Add – Mean Factor of 8 +/- 2
  – Reconfigurable IP Multiply and Add – Mean Factor 4 +/- 1
  – Bigger fixed point DSP’s – Mean Factor of 3

• Benefit
  – More Floating Point ALU’s
  – More routing paths
Number of AE FPGA’s per node

• Feature Set
  – 4, 8, or 16 as a function of physical memory capacity
    • 4 - One byte per flop – mean factor of 1
    • 8 - 1/2 byte per flop – mean factor of 2
    • 16 - 1/4 byte per flop – mean factor of 4

• Benefit
  – More Internal Parallelism
  – Transparent to user
  – Potential heterogeneity within node
Convey Compilers

- Program in ANSI standard C/C++ and Fortran
  - PGAS ready
- Unified compiler generates x86 & coprocessor instructions
- Seamless debugging environment for Intel & coprocessor code
- Executable can run on x86_64 nodes or on Convey Hybrid-Core nodes
Programming Model

Example 4.1-1: Matrix by Vector Multiply

1: #include<upc_relaxed.h>
2: #define N 200*THREADS
3: shared [N] double A[N][N];   NOTE: Thread is 16000
4: shared double b[N], x[N];
5: void main()
6: {
7: int i,j;
8: /* reading the elements of matrix A and the
9: vector x and initializing the vector b to zeros
10: */
11: upc_forall(i=0;i<N;i++;i)
12: for(j=0;j<N;j++)
14: }
Math results in

• Using the mean
  – 7 Moore’s law
  – 4 Matrix arithmetics
  – .90 efficiency (percentage of peak)
  – 8 Fused multiply/add (64 bits)
  – 4 16 AE’s (user visible pipelines) per node
  – Or a MEAN of 800 times today
    • Best Case – 2304
    • Worst Case - 448
The System

- 2010 base level node is 80 GFlops/node (peak)
- Thus $7 \times 4 \times 0.9 \times 8 \times 4 = 806$ Factor
  - Mean of 800 = +1500 (upside)/-400 (downside)
  - 64 TFlops/node (peak)
  - 16,000 Nodes/Exascale Linpack
- 64 bit virtual address space
  - Flat address space
  - UPC addressing paradigm integrated within TLB hardware
  - Programming model is 16000 shared memory nodes
    - Compiler optimizations (user transparent) deal with local node micro-architecture
- Power is 2 KWatts/Node (3U rack mounted)
  - 32 MegaWatts/system
  - 32 TBytes/Node (288 PetaBytes – system)
  - Physical Memory approx 60% of power
INTEGRATED SMP - WDM

DRAM – 16/32 TeraBytes - HIGHLY INTERLEAVED

MULTI-LAMBDA xmit/receive

CROSS BAR

6.4 TBYTES/SEC

.1 bytes/sec per Peak flop

swallach – CARL – Micro 43
IBM Optical Technology

COTS ExaFlop/s System

Single Node
16 FPGA AE

ALL-OPTICAL SWITCH

I/O

LAN/WAN

10 meters = 50 NS Delay

swallach – CARL – Micro 43
What can be done to:

• Reduce power
  – 3D packaging
    • Reduce power in chip to chip interface
    • Better DRAM utilization
    • More power domains (selectively power up/down die regions)
• Architecture (relative to strawman)
  – Less Nodes
    • More matrix arithmetics
      – Twice the performance, same power
    • More floating point IP
      – Twice the performance, same power
  – Less Physical Memory
    • ¼ byte per flops or 15 TBytes/Node yields 20 MWatts
    • (144 PetaBytes Total)
Concluding

- Uniprocessor Performance has to be increased
  - Heterogeneous here to stay
  - The easiest to program will be the correct technology
- Smarter Memory Systems (PIM)
- New HPC Software must be developed.
  - SMARTER COMPILERS
  - ARCHITECTURE TRANSPARENT
- New algorithms, not necessarily new languages

"He's working his way up to saying something inspirational."
Finally

I HATE SALES. CAN YOU CROSS-TRAIN ME TO BE AN ENGINEER?